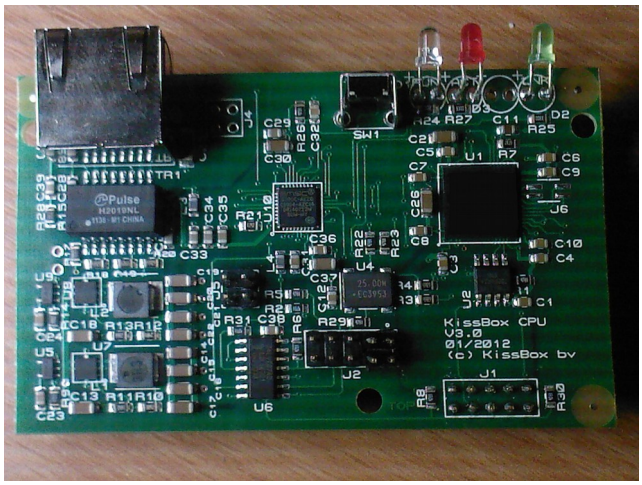
KISS-BOX

APPLICATION NOTE AN-003

ADDING RTP-MIDI/HD TO ARDUINO



Contents

1 - Introduction

The KissBox RTP-MIDI OEM board is a specialized network processor in charge of handling the whole RTP-MIDI communication stack and provide directly MIDI and HD data to an external application board.

Arduino is powerful prototyping and development ecosystem, with native MIDI capabilities. However, most Arduino board have very limited RAM, which can make native RTP-MIDI implementation rather hard. Moreover, most Arduino will require an external Ethernet shield, which has negative performance impact for small micro-controllers.

RTP-MIDI OEM board is an ideal partner for Arduino platforms, since it takes in charge the complete protocol stack and frees the whole CPU power for user application. Moreover, the RTP-MIDI OEM board can buffer significant amount of data thanks to its amount of RAM. So the whole RAM of the Arduino is available for the user application. It also relaxes response time constraints on user's program (there is no need to implement interrupts to avoid data losses, the data are buffered in the RTP-MIDI OEM board if necessary)

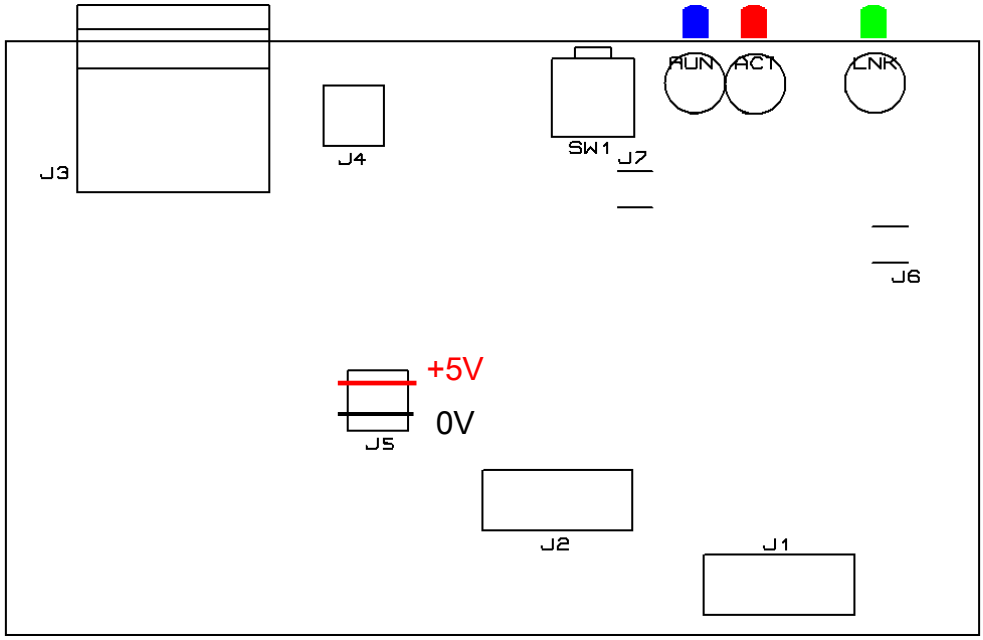
The network CPU can be used both in MIDI and HD setups. Change from MIDI to HD (and back) is simply done by updating the board firmware (Note : access to HD firmware is restricted to MMA HDWG members until official publication of HD protocol specification)

This application note describes how to interconnect the RTP-MIDI OEM board with various Arduino boards (Uno, Mega2560, Duo, Galileo Gen 1 and Gen 2, etc...)

2 - RTP-MIDI OEM board characteristics

- Dual core 32 bits CPU running at 400MHz
- 10/100 Mbps Ethernet with MDIX and auto-negotiation support
- 128 Kb on-chip fast SRAM for code and data
- 4 Mbits Flash for bootloader, file system and firmware storage
- PoE option (via daughterboard)
- Power supply : 5V (can be shared with Arduino board)
- Power consumption : typically 150mA under 5V
- Full RTP-MIDI stack included in the module, with complete RTP-MIDI endpoint implementation
- Up to four parallel sessions (with automatic merging / MIDI THRU)
- "Bonjour" support for automatic discovery
- Static IP / DHCP / Zeroconf supported
- Fast SPI communication with Arduino (up to 10MHz). No tricky "high speed serial port" needed.

3 - RTP-MIDI OEM hardware interface



3.1 - J2 – Connector to host device

Connector J2 is used to connect RTP-MIDI OEM board to the Arduino (for 3.3V Arduinos) or to the 3.3V adapter board (for 5V Arduinos). This connector can also be used to provide power supply to the CPU board if you do not want to use J5.

All lines use 3.3V signaling levels and are NOT 5V TOLERANT. **They shall not be used to connect directly to 5V Arduino boards. A level translator is required in this case.**

Connector is a 2x5 pins header, 0.1" spacing.

The table hereafter list the different signals used for the SPI communication.

Pin#	Function	Direction	Description
1	I/O 1	Input	Reserved for future use
2	I/O 2	Input	Reserved for future use
3	+5V		+5V power supply to/from Arduino. This pin is connected to pins 1 and 2 of J5. The pin can be used to supply 5V to the RTP-MIDI OEM from the Arduino. When a power supply is connected to J5, this pin can be used to power the Arduino connected to J2 Voltage on this line **must** be kept under 5.25V under any circumstance. A voltage greater than 5.25V applied to this line even for a short time can destroy the RTP-MIDI OEM board
4	GND		0V connection for power-supply
5	nRESET	Input	OEM board reset. Forces the RTP-MIDI OEM board to reset when a logic "0" is applied for at least 10 milliseconds. The board is reset when the nRESET signal is de-asserted (rising edge). The input has an integrated 4k7 pull-up resistor.
6	nSPICS	Input	SPI Chip Select – Active low
7	SCLK	Input	SPI Serial Clock.
8	I/O 4	Input	Reserved for future use
9	MISO	Output	Serial Data Output. Data from OEM to Arduino. WARNING: this output does not go into high-impedance mode when nSPICS is de-asserted! An external tristate buffer is required when the SPI is shared between RTP-MIDI OEM and other SPI devices.
10	MOSI	Input	Serial Data Input. Data from Arduino to RTP-MIDI OEM.

3.2 - J5 – 5V power supply

Used to provide 5V power supply to the CPU board, from PoE daughterboard or from external power supply if PoE option is not installed.

WARNING : the power supply voltage shall ALWAYS be kept between 4.75V and 5.25V

The board is not guaranteed to work properly if voltage drops under 4.75V

The board can be damaged/destroyed if voltage applied to J5 exceeds 5.25V, even for a short time.

Connector is a 2x2 pins header, 0.1" spacing.

Pin#	Function	Description
1	+5V	+5V power supply
2	+5V	+5V power supply
3	GND	0V
4	GND	0V

3.3 - Reserved connectors

Connectors J1, J4, J6 and J7 are reserved for special uses and future expansions and shall not be connected to Arduino.

4 - Connecting the RTP-MIDI OEM to Arduino

4.1 - Connection to 3.3V Arduino boards

IMPORTANT: **Never connect 5V Arduino boards directly to KissBox RTP-MIDI OEM board!** This would damage/destroy the OEM board! This chapter only applies to 3.3V Arduino.

At the time this document is written, only the following Arduino boards have been validated for direct connection to the RTP-MIDI OEM:

- Intel Galileo Gen 1
- Intel Galileo Gen 2
- Arduino DUE

VERY IMPORTANT: the Intel Galileo boards can work both with 5V and 3.3V shields. By default, Intel delivers the board configured for 5V. **So you must select the 3.3V mode before connecting the RTP-MIDI OEM board to the Galileo**. To do that, move the IOREF jumper on 3.3V position (see photo 1).

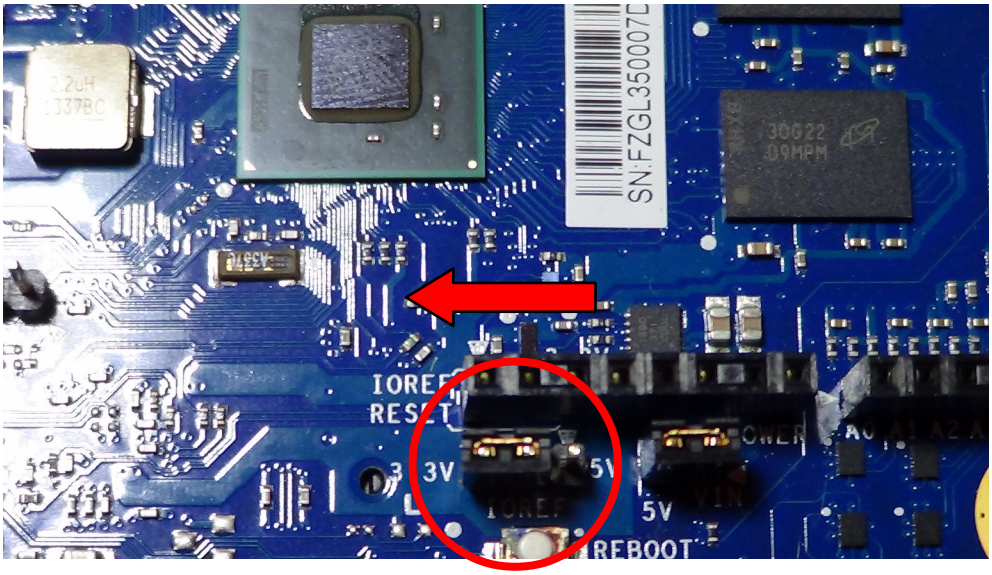


Photo 1: Position of 3.3V jumper on Intel Galileo

If you have any doubt about the I/O voltage of your Arduino, check the voltage between IOREF pin and ground on your Arduino. If you see a voltage of 3.3V, then you can connect your Arduino directly to the KissBox RTP-MIDI OEM board. If you see a voltage of 5V, then you **must use a voltage adapter** between the Arduino and the RTP-MIDI OEM (see chapter 4.2).

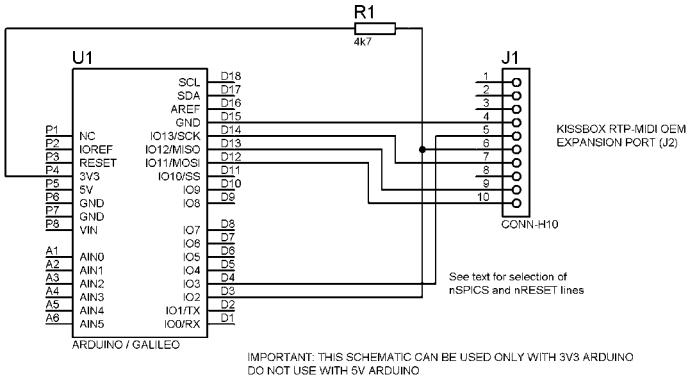


Diagram 1: Connection diagram for 3.3V Arduino boards

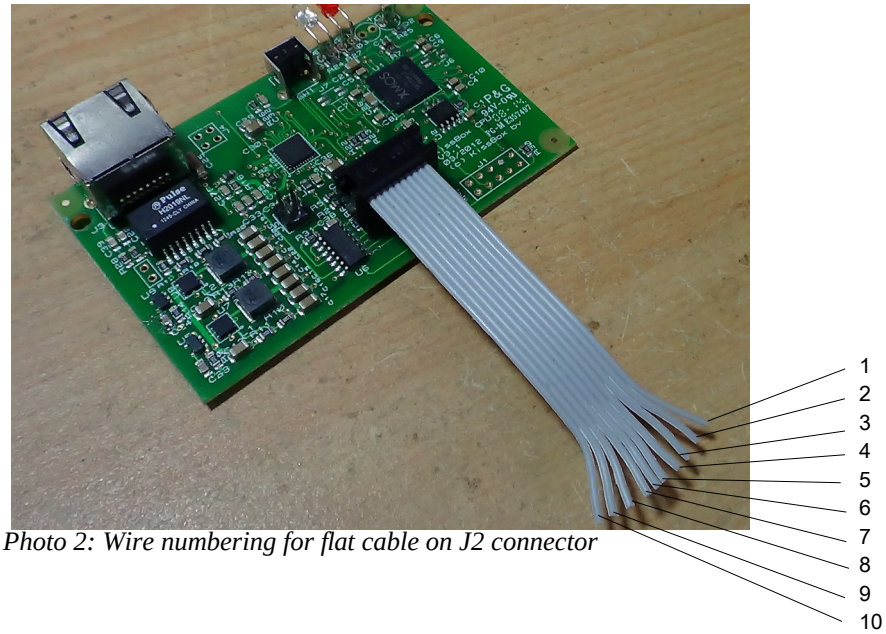


Photo 2: Wire numbering for flat cable on J2 connector

4.2 - Connection to 5V Arduino boards

The RTP-MIDI OEM module only supports 3.3V voltage levels on J2. It shall **never** be connected directly to a micro-controller system using 5V voltage level, otherwise the microprocessor on the RTP-MIDI OEM board will be damaged.

A voltage translator is then required between 5V Arduino boards (like UNO or MEGA2560 boards) and the RTP-MIDI OEM board. There are multiple schematics available on the web for such translators.

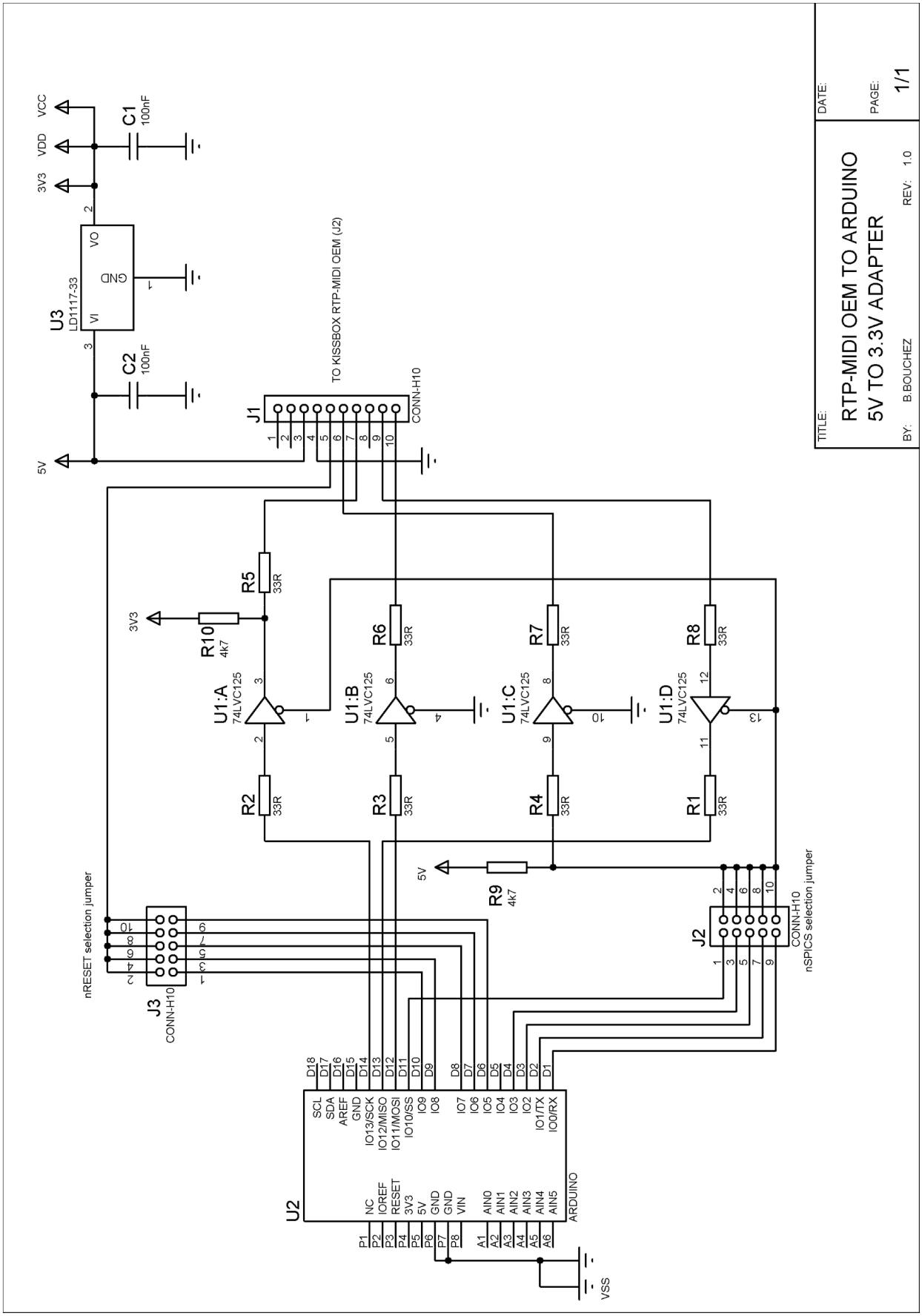
Many translators are based on CD4050 chips. Note that these chips are rather slow when powered on 3.3V (the propagation delay is around 150ns in this case). If you use a CD4050 based voltage translator, please note that you will probably need to lower the SPI clock frequency, otherwise you may encounter communication errors between the Arduino and the RTP-MIDI OEM module.

We recommend the use of the 74LVC125 which provides excellent propagation delays (less than 10ns). However, these chips are not available in DIP package (only SO14), which can make them hard to solder for amateurs.

It is also possible to use 74LVC07 chip, but do not forget that they have open collector outputs, and they require pull-up resistors to operate (1k pull-ups are sufficient to provide good switching times)

You can find hereafter the schematic of the voltage translator we recommend to use to interconnect the RTP-MIDI module with an Arduino. This voltage translator is available fully assembled and tested from KissBox and BEB (http://imodularsynth.com) as a spare part and included in the Arduino RTP-MIDI kits. This voltage translator

includes also a MISO separation buffer (see next paragraph) which allows to share the SPI bus with other SPI devices. Note that the voltage translator can also be used with 3.3V Arduino boards.



4.3 - Sharing SPI with other devices

The RTP-MIDI OEM board does not have tri-state output buffers on the MISO line. This line remains active when nSPICS signal is de-asserted. This prevents the SPI bus to be shared with other SPI devices when the RTP-MIDI OEM board is connected.

If you intend to connect only a RTP-MIDI OEM board to the SPI lines of your Arduino (so the SPI bus is not shared between the RTP-MIDI OEM and other devices), there is no need for extra circuitry.

If you intend to connect RTP-MIDI OEM board and other SPI devices at the same time, you will need to add a tristate buffer (typically a 74HCT125) between the MISO line of OEM board and the MISO line of the Arduino. This buffer shall be controlled by the nSPICS signal going to the RTP-MIDI OEM, as described in the following schematics:

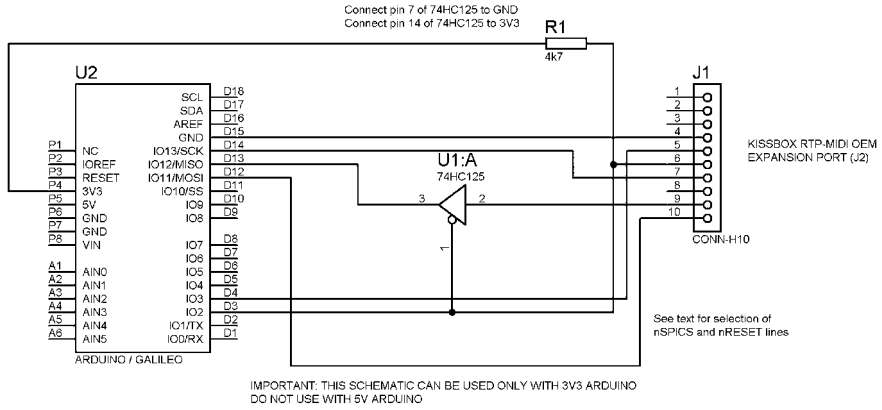


Diagram 2: Add a tri-state buffer if SPI is shared with multiple devices (example given here for 3.3V Arduino. The adapter for 5V Arduino includes the tri-state buffer)

4.4 - Connection to the Arduino I/O lines

The RTP-MIDI OEM (either directly or through the level adapter) must be connected to SPI lines of the Arduino. The SPI lines are located on the following pins on most Arduino boards (Uno, Mega2560, Due, Galileo, etc...) :

Function	Arduino	RTP-MIDI J2
MOSI	IO11	10
MISO	IO12	9
SCLK	IO13	7
nSPICS	See text	6
nOEMRESET	See text	5
Ground (0V)	GND	4

If you have a different Arduino model, check the schematics/user's manual to check the SPI location. Note that some Arduino board do not support SPI. These boards can not be used with the RTP-MIDI OEM board.

4.4.1 - SPI selection signal

A SPI device also needs a selection line (nSPICS) which indicates when the device is selected for communication with the SPI master. This signal is a simple digital output, controlled directly from your script. If you have multiple SPI devices connected to the same Arduino, you have to assign one selection line (so one digital output) per device. The other SPI lines are shared between all SPI devices.

For the Intel Galileo, it is highly recommended to use IO2 or IO3 for the nSPICS signal, in order to get the best possible performances on SPI. These lines are controlled directly by the processor, not by the IO expander, so they are much faster to react (around 100 microseconds with IO2 or IO3, around 2 milliseconds with others IO lines). And if you use IO2 or IO3, do not forget to use OUTPUT_FAST rather than OUTPUT.

For all other Arduino boards, you can use any I/O line for nSPICS as long as it is not used for another function.

It is highly recommended to place a 4.7kohms resistor between the nSPICS line and 3.3V power supply, as described in the schematics. Note that this resistor is part of the 5V->3V3 adapter, so you do not need to install it yourself for any 5V Arduino.

4.4.2 - RTP-MIDI board reset

The RTP-MIDI OEM board has been extensively tested with as many different Arduino as possible. During these tests, we have seen that it was possible to get illegal SPI sequences (especially on Galileo during the boot time or when a sketch is being reloaded). This especially happens when Arduino takes a very long time to start, which put the SPI lines in undefined states while the RTP-MIDI OEM board has already started.

These abnormal SPI sequences can lead to a corruption in the input/output shift register of the RTP-MIDI board (the symptom is a shift on the right or on the left of the 32 bits word exchanged with the RTP-MIDI OEM).

In order to guarantee the synchronization between the Arduino and the RTP-MIDI OEM, a reset line has been added. The Arduino can then inform the RTP-MIDI OEM board that it is (re)starting, which will force the RTP-MIDI board to restart too.

It is recommended to add a reset sequence in the setup() function, so the RTP-MIDI board is guaranteed to be reset when the sketch is starting. Please refer to the code below for a reference implementation of the reset function.

5 - SPI configuration

The RTP-MIDI OEM module uses the following SPI configuration :

- SPI Mode 3 (CPOL = 1 / CPHA = 1)
- 32 bits per word
- MSB transmitted first

The KissBox RTP-MIDI OEM accepts a clock frequency up to 10MHz. Please note that SPI signals becomes sensitive to cabling when the clock frequency is high. If you experience communication errors, just lower the SPI clock frequency. There is no minimum frequency.

As a reminder for SPI clock divider :

SPI_CLOCK_DIV2 = 8MHz SPI clock

SPI_CLOCK_DIV4 = 4MHz SPI clock

SPI_CLOCK_DIV8 = 2MHz SPI clock

```
/*
Minimum configuration data required in setup() for communication with RTP-MIDI OEM
board from KissBox
*/

#include <SPI.h>

#define nSPICS    3            // IO3 is used here for nSPICS. Change pin number
depending on your own configuration

void setup()
{
  // It is recommended to initialize the nSPICS output as fast as possible after
startup to avoid the RTP-MIDI OEM module to see a floating line which would lead to
false SPI synchronization
// Prepare pin for CS signal
  //pinMode (nSPICS, OUTPUT);     // Use this line on all Arduino except Galileo
  pinMode (nSPICS, OUTPUT_FAST);  // Use this line on Galileo
  digitalWrite (nSPICS, HIGH);    // Set SPICS to inactive state

  // Configure the SPI interface for RTP-MIDI OEM
  SPI.begin();
  SPI.setBitOrder (MSBFIRST);
  SPI.setDataMode (SPI_MODE3);
  SPI.setClockDivider (SPI_CLOCK_DIV8);          // 2 MHz clock signal.
}
```

6 - MIDI over SPI communication protocol

MIDI communication protocol is based on MMA Specification for USB communication.

MIDI data are carried as 32 bit MIDI Event. Most common MIDI messages are 2 or 3 bytes packed into one MIDI Event. Longer messages (System Exclusive) messages are carried in multiple MIDI Events. These MIDI Event provide a method to transfer MIDI messages with 32 bit fixed length messages to help memory allocation. This also makes parsing MIDI events easier by packetizing the separate bytes of a MIDI message into one parsed MIDI Event.

The first byte in each 32-bit MIDI Event is a sub-header containing a Session Index Number (SIN) on 4 bits followed by a Code Index Number (CIN) on 4 bits. The remaining three bytes contain the actual MIDI event. Most typical parsed MIDI events are two or three bytes in length. Unused bytes are reserved and must be padded with zeros (in the case of a one- or two-byte MIDI event) to preserve the 32-bit fixed length of the MIDI Event.

Byte 1		Byte 2	Byte 3	Byte 4
SIN	CIN	DATA_1	DATA_2	DATA_3

6.1 - Session Index Number

Session Index Number is used to identify the RTP-MIDI session handler providing MIDI data. Multiple MIDI streams can then use the same SPI link.

6.2 - Code Index Number

The Code Index Number (CIN) indicates the classification of the bytes in the DATA_X fields and the number of bytes in the message. The following table summarizes these classifications.

CIN	DATA_X size	Description
0x0	1, 2 or 3	Miscellaneous function codes. Reserved for future extensions.
0x1	1, 2 or 3	Cable events. Reserved for future expansion.
0x2	2	Two-byte System Common messages like MTC, SongSelect, etc.
0x3	3	Three-byte System Common messages like SPP, etc.
0x4	3	SysEx starts or continues
0x5	1	Single-byte System Common Message or SysEx ends with following single byte
0x6	2	SysEx ends with following two bytes.
0x7	3	SysEx ends with following three bytes.
0x8	3	Note-off
0x9	3	Note-on
0xA	3	Poly-KeyPressure
0xB	3	Control Change
0xC	2	Program Change
0xD	2	Channel Pressure
0xE	3	PitchBend Change
0xF	1	Single-byte message

6.3 - DATA_1, DATA_2, DATA_3: MIDI Message Data

MIDI data bytes are transmitted in DATA_1, DATA_2 and DATA_3 exactly in the same format as the MIDI 1.0 specification requires.

Running status is never used, so all the messages are formed of all bytes.

6.4 - 32 bits messages examples

All examples hereafter are given with Session Index Number = 0

6.4.1 - Timing clock

MIDI message = 0xF8

SPI message = 0x0FF80000

6.4.2 - Note On

MIDI message = 0x90 0x2A 0x40

SPI message = 0x09902A40

6.4.3 - System Exclusive

MIDI message = 0xF0 0x01 0x02 0x03 0x04 0x05 0xF7

SPI messages = 0x04F00102 (SYSEX start)

0x04030405 (SYSEX continues)

0x05F70000 (SYSEX ends with one byte)

6.5 - "Stuffing" messages

Since the SPI link exchanges input and output data at the same time, MIDI data will not be available from both side most of the time. For example, when a MIDI message is being received from the network, it's highly probable that the Arduino has no data to send at the same moment.

In order for the receiver (on each side) to detect that there was no MIDI data available in one direction, two reserved values are used to report the lack of data:

- 0xFFFFFFFF
- 0x00000000

RTP-MIDI OEM will send one of these values when a SPI transfer is initiated and no data are available in its buffers. The Arduino shall reject the 32 bits message received from the RTP-MIDI OEM if it contains one of these values.

In the other direction, when the RTP-MIDI OEM receives this value from the Arduino, it will recognize the reserved value as "No data available". The data is then ignored and not transmitted over the network.

7 - A first Arduino program to test your setup

Once everything is connected, you can use the following sketch to test the whole setup. This sketch simply displays any MIDI data received by the RTP-MIDI OEM.

You will need another RTP-MIDI client on the other side, to send data to the RTP-MIDI OEM. This can be:

- PC with rtpMIDI driver installed
- Mac computer (RTP-MIDI driver is a part of Mac OS since 10.4)
- iPad/iPhone running a MIDI application (RTP-MIDI is included in iOS since version 4.2)
- a KissBox RTP-MIDI interface
- etc...

We recommend you to look at the following PDF document, available on KissBox website "**RTP-MIDI Integration guide for Windows and MacOS**". This document describes in detail how to create a RTP-MIDI setup on PC and Mac computers.

If you use an iPad, then you can use the following document from KissBox "**RTP-MIDI Integration guide for iOS devices**", also available freely on KissBox website.

```
/*
Test sketch for RTP-MIDI OEM
Original version by Benoit BOUCHEZ - 2015
The following code is public domain.

This sketch displays the incoming data from KissBox RTP-MIDI OEM board.
Before using it, make sure that the IO line used for nSPICS signal is correctly
selected (depends on your setup)
For Galileo Gen 1 or Gen 2, it is highly recommended to use IO2 or IO3 configured as
OUTPUT_FAST to get the best possible speed

Report to KissBox "RTP-MIDI Integration Guide for Windows and MacOS" document to
know how to start a RTP-MIDI client on your PC or Mac
Any MIDI data sent from PC or Mac will be displayed in the Arduino console. ENJOY!
*/

#include <SPI.h>

#define nSPICS    2   // IO2 is used here for nSPICS. Change pin number depending on
your own configuration

// Global variables
unsigned char ByteFromRTPMIDI[4];

void setup()
{
  Serial.begin (9600);    // For debugging purposes

  pinMode (nSPICS, OUTPUT);       // Prepare pin for CS signal
  digitalWrite (nSPICS, HIGH);    // Set SPIDCS to inactive state

  // Configure the SPI interface for RTP-MIDI OEM
  SPI.begin();
  SPI.setBitOrder (MSBFIRST);
  SPI.setDataMode (SPI_MODE3);
  SPI.setClockDivider (SPI_CLOCK_DIV4);
}  // setup

```

```
// -------------------------

void loop()
{
  int x;

  // Get 32 bits word from the Arduino
  digitalWrite (nSPICS, LOW);
  for (x=0; x<4; x++)
  {
    ByteFromRTPMIDI[x]=SPI.transfer(0xFF);    // Sending 0xFFFFFFFF will inform the
RTP-MIDI OEM that we have nothing to send
  }
  digitalWrite (nSPICS, HIGH);

  /*
    When we receive 0x00000000 or 0xFFFFFFFF from RTP-MIDI OEM, it means that we
have received nothing from RTP-MIDI
    Since there is no message starting with 0x00 (reserved) or 0xFF (single byte
message from session 0xF, which does not exist with current firmware),
    we can make an easy filter with just these two values to be checked
  */
  if ((ByteFromRTPMIDI[3]!=0x00)&&(ByteFromRTPMIDI[3]!=0xFF))
  {
    // We print in reverse order, to make it easier to read
    for (x=3; x>=0; x--)
    {
      Serial.print (ByteFromRTPMIDI[x], HEX);   // Display the bytes sent by the
RTP-MIDI OEM
      Serial.print (" ");
    }
    Serial.println ("");
  }
}  // loop
```

IMPORTANT : we recommend you strongly to restart the Arduino and the RTP-MIDI OEM (by switching them OFF then ON) each time you have loaded a new sketch. This will force the synchronization between the two boards.

If you see garbled data on the terminal every time you receive a MIDI message, simply reset the two boards. This happens when the SPI is not correctly synchronized (due to an incomplete SPI sequence from the Arduino, when it is reset in the middle of a transfer)

Here is an example of what you can get on the serial console :

B B0 0 0

B B0 20 0

C C0 26 0

And here is how to interpret this:

B B0 0 0 : CIN = B, Data1 = 0xB0 : Control Change received on Channel 0 by RTP-MIDI session number 0, Control number = 0, Control value = 0

B B0 20 0 : CIN = B, Data1 = 0xB0 : Control Change received on Channel 0 by RTP-MIDI session number 0, Control number = 32 (0x20), Control value = 0

C C0 26 0 : CIN = C, Data1 = 0xC0 : Program Change received on Channel 0 by RTP-MIDI session number 0, Program = 38 (0x26). Last byte is ignored (0) since Program Change message is only 2 bytes on MIDI

8 - HD communication protocol

HD communication protocol is based on 32 bits words, making a complete HD message a multiple of 32 bits.

Compared to MIDI 1.0, HD does not have a specific marker to identify the first word of a message. In order to allow the host CPU to synchronize properly on any new incoming HD message, HD firmware for RTP-MIDI OEM uses a dedicated synchronization protocol.

Details for HD communication protocol are restricted for now to MMA HD Working Group members until official release of specification.

If you want to implement HD communication protocol in your host, please contact Kiss-Box to provide your HDWG identity. We will then provide the details of the HD communication protocol under NDA.

9 - Document revisions

Date	Auteur	Version	Description
10/09/2015	B.Bouchez	1.0	First draft

Prepared with OpenOffice Writer software.

For more information about OpenOffice, go to http://www.openoffice.org.